When we have to create two or more Factory classes, then we can use Abstract Factory pattern for this

Step 1: Create Interface for our factories

This interface will have a definition which will return created object

interface IFactory

{

FactoryDataItem GetData(int id);

}

Create Class

public FactoryDataItem(object dataItem)

{

\_dataItem = dataItem;

}

object \_dataItem;

public object DataItem

{

get

{

return \_dataItem;

}

}

}

Create Abstract Class , implement Interface

abstract class AbstractFactory : IFactory

{

public abstract FactoryDataItem GetData(int type);

}

We are not providing any details here ,

#region Shapes

enum ShapeType { Circle = 1, Rectangle = 2, Triangle = 3 };

class Circle

{

public void GetInfo()

{

Console.WriteLine("This is a Circle");

}

}

class Rectangle

{

public void GetInfo()

{

Console.WriteLine("This is a Rectangle");

}

}

class Triangle

{

public void GetInfo()

{

Console.WriteLine("This is a Triangle");

}

}

#endregion

#region Colours

enum ColourType{ Red=1, Green=2,Yellow=3};

class Red

{

public void GetInfo()

{

Console.WriteLine("This is a Red Colour");

}

}

class Green

{

public void GetInfo()

{

Console.WriteLine("This is a Green Colour");

}

}

class Yellow

{

public void GetInfo()

{

Console.WriteLine("This is a Yellow Colour");

}

}

#endregion

Create Factory Classes now

class ShapeFactory : AbstractFactory

{

public override FactoryDataItem GetData(int type)

{

FactoryDataItem factoryDataItem = null;

switch((ShapeType) type)

{

case ShapeType.Circle :

factoryDataItem = new FactoryDataItem(new Circle());

break;

case ShapeType.Rectangle:

factoryDataItem = new FactoryDataItem(new Rectangle());

break;

case ShapeType.Triangle:

factoryDataItem = new FactoryDataItem(new Triangle());

break;

}

return factoryDataItem;

}

}

class ColourFactory : AbstractFactory

{

public override FactoryDataItem GetData(int type)

{

FactoryDataItem factoryDataItem = null;

switch ((ColourType)type)

{

case ColourType.Red:

factoryDataItem = new FactoryDataItem(new Red());

break;

case ColourType.Green:

factoryDataItem = new FactoryDataItem(new Green());

break;

case ColourType.Yellow:

factoryDataItem = new FactoryDataItem(new Yellow());

break;

}

return factoryDataItem;

}

}

class Program4

{

static void Main()

{

ShapeFactory shapeFactory = new ShapeFactory();

ColourFactory colourFactory = new ColourFactory();

Circle circle = shapeFactory.GetData((int)ShapeType.Circle).DataItem as Circle;

circle.GetInfo();

Red red = colourFactory.GetData((int)ColourType.Red).DataItem as Red;

red.GetInfo();

}

}